GrabBag – A Data Structure for Non-Deterministic Element Selection in C++11

(Version 1.0.0)

Daniel Kostuj

July 17, 2018

1 Introduction

Imagine, in front of you: a paper bag, filled with objects of the same type that may or may not have different attributes. A *grab bag*, if you will.

Without looking into the bag, thrust your hand into it. If you can feel that the bag contains these objects, grab such an object and take it out of the bag.

Now you can finally get a look at the object you picked. Only then do you truly know the selected object's attributes.

The data structure `GrabBag<T>` tries to approximate this behavior, using a generic data type implementation.

Essentially, it is a wrapper class for a `std::vector<T>` with its own methods, allowing for effective modularization.

Under the hood, this class uses C++11 STL methods for full compability with C++11 and onwards.

Should the need for a non-deterministic element selection arise: Pick `GrabBag`.

2 Installation

The class `GrabBag` is fully implemented in the header file `GrabBag.h` inside the `src` folder. Just copy this header file into your local project or global C++ library folder to use the `GrabBag` functionalities. Usage examples are described in further detail in the next sections.

In addition to the class definition, the folder `src` contains a unit test suite file `UnitTests.cpp`. These unit tests are run every time a new commit is completed, however you can also run the unit tests by executing the `launchUnitTests.sh` script.

I'll misuse this space to thank the group behind the single-header unit testing framework Catch2 for their great work.

3 Usage Examples

3.1 Lottery

The most obvious `GrabBag` use case would be implementing a lottery program, where a fixed amount of numbers are drawn from a number pool.

The following example is a program that simulates the Italian *SuperEnalotto* lottery.

In this particular lottery, seven numbers are drawn out of a pool of 90 numbers. Also, a "super star" number is drawn out of a separate 90-number pool[1].

```
#include "../../src/GrabBag.h"
#include <iostream>
#include <set>

int main() {
    GrabBag<int> enaLottoPool;
    GrabBag<int> superStarPool;
    std::set<int> mainNumbers;
    int jolly, superStar;

    for (int i = 1; i <= 90; ++i) {
        enaLottoPool.insert(i);
        superStarPool.insert(i);
    }

    for (int j = 0; j < 6; ++j)
        mainNumbers.insert(enaLottoPool.grab());

    jolly = enaLottoPool.grab();
    superStar = superStarPool.grab();

    // Output Main Numbers
    std::cout << "Main Numbers: ";
    for (int number : mainNumbers)
        std::cout << number << " ";
    std::cout << std::endl;

    // Output Jolly & SuperStar numbers
    std::cout << "Jolly: " << jolly << std::endl;
    std::cout << "SuperStar: " << superStar << std::endl;
}
```

Example output:

```
$ ./a.out
Main Numbers: 18 25 35 43 68 87
Jolly: 6
SuperStar: 8
```

[1] An example of the lottery in action can be watched here.

3.2 Permutation

A `GrabBag` can be used to randomly permutate an array-like structure of objects, as shown in the next example program.

The behavior of this program is reminiscent of the Fisher–Yates shuffle algorithm, also known as Knuth's Algorithm P.

```
#include "../../src/GrabBag.h"
#include <iostream>
#include <string>

int main(int argc, char **argv) {
    if (argc == 1) {
        std::cerr << "Need a string argument" << std::endl;
        return 1;
    }

    // Program argument parameter is input string
    std::string input(argv[1]);
    std::string output;
    GrabBag<char> word;

    // Insert single char from input string into GrabBag
    for (char c : input)
        word += c;

    // Fill output string with random char
    while (!word.empty())
        output += word.grab();

    std::cout << output << std::endl;
}
```

Example output:

```
$ for i in {1..20}; do ./a.out Permutation; done
meunioattrP
tioPrtmeuna
amtireoPtnu
Piartuoemtn
utamnitPreo
unrttiaPome
rPnutaieotm
oemPtatunir
ainotrPumet
iatenrtuoPm
aPiumeonttr
tonmiaPuert
aimPettunro
iuaorntetmP
ePmnrtutaio
uonrmPetati
Ptotimeuarn
emrituPaont
tnrtmoiPuae
oPetmratinu
```

3.3 Reservoir Sampling

4 Methods

Here is a short overview of all methods defined inside the GrabBag, including small examples.

4.1 `GrabBag<T>()`

Instantiates an empty `GrabBag` that accepts elements of type `T`. Size of `GrabBag` will be set to zero.

4.2 `GrabBag<T>(const std::vector<T> &newBag)`

Instantiates a `GrabBag` and assigns its content to the vector `newBag<T>`. Size of `GrabBag` will be set to the size of vector.

4.3 `GrabBag<T>(const GrabBag<T> &secondBag)`

Copy constructor

4.4 `void insert(T object)`

Inserts the `object` of type `T`, preferably into a `GrabBag` of type `T`.

4.5 `void insert(const std::vector<T> &objects)`

Appends the content of vector `objects` to the `GrabBag` elements.

4.6 `T grab()`

Returns an element from the `GrabBag` at a random index. Afterwards, the `Grabbag` will not contain that element anymore.

4.7 `std::size_t size()`

Returns the amount of elements that take up the `GrabBag` instance.

4.8 `bool empty()`

Return `true` if `size()` equals 0; otherwise, `false`.

4.9 `void operator+=(T e)`

A fancy way to call `insert(T object)`.
Syntactic sugar.

4.10 `void operator+=(const std::vector<T> &vec)`

A fancy way to call `insert(const std::vector<T> &objects)`.
Again, syntactic sugar.

5 License